Mechanistic Miscomputation

Mechanistic miscomputation: a reply to Fresco & Primiero

Fresco & Primiero are certainly correct to note that despite the attention that computation has received in recent decades, there has been a distinct failure to account for miscomputation (2013: 2). The analysis and taxonomy that they provide is in this sense a welcome conversation-starter. Of particular importance is their distinction between operational/function errors and design/conclusion errors. I will briefly review this distinction, before arguing that according to the mechanistic account of computation (as presented in Piccinini 2007), only operational/function errors count as genuine cases of miscomputation. In closing I will consider whether this restriction should be seen as a strength or a weakness of the mechanistic account.

Miscomputation, according to Fresco & Primiero, can occur at any level of abstraction, ranging from the conceptual, through the algorithmic, to the physical (2013: 6-14). Examples of each can be found in their article. Of particular relevance here is the distinction that they draw between those errors that depend upon the intentions or actions of a designer, and those that are merely the result of a faulty mechanism or component. Drawing on Turing (1950: 449), they refer to the former as "errors of conclusion" and the latter as "errors of function" (Fresco & Primiero 2013: 2-3). They also refer to these as "design errors" and "operational malfunctions" respectively (*ibid*: *passim*). Here I will primarily use the latter terminology, as I feel it better captures the spirit of the distinction.

Design errors occur whenever an agent involved in the creation of a piece of hardware or software fails to correctly implement their vision of how the component should function. Operational malfunctions occur whenever this implementation, regardless of its intended purpose, breaks down in some way. Fresco & Primiero's taxonomy includes both as cases of miscomputation (2013: 15).

As Fresco & Primiero acknowledge, Piccinini's mechanistic account emphasises the need for any theory of computation to account for the possibility of miscomputation (Piccinini 2007: 505; Fresco & Primiero 2013: 2). Piccinini lists "hardware failure […] mistake in computer design […] programming error […] round off errors [and] faulty interaction between hardware and software" as potential kinds of miscomputation (2007: 523-4). Fresco & Primiero rightly draw attention to the fact that this list conflates operational malfunctions with design errors (2013: 2).

As I will go on to argue, only the first (and perhaps the last) should be considered genuine cases of miscomputation under the mechanistic account of computation. What I mean by this is that

the mechanistic account should recognise only operational malfunctions, but not design errors, as cases of miscomputation.

A computing mechanism is defined as a physical system composed of strings of digits and a processor (or processors) that transform those strings (Piccinini 2007: 512-4). In addition it is likely to possess non-computational components such as batteries, and computational components such as input and output devices, although such additional components are not strictly required (*ibid*: 514-6).

A mechanical fault in any of these components that changes the expected behaviour of the system will qualify as an operational malfunction, causing the computing mechanism to miscompute. It is possible that a mechanical fault will not have any impact on the expected behaviour, and in these cases there will be no miscomputation (cf. Fresco & Primiero 2013: 265). A fault that does change expected behaviour is identified by the failure of the system to perform as designed. That is to say, a component malfunctions if it fails to implement the correct function, and a computing mechanism miscomputes if it fails to perform the correct computation.

There is some ambiguity as to whether we should be concerned only with external (i.e., input/output) behaviour, or with both internal and external behaviour. Taken to an extreme, the latter position could imply that any internal mechanical fault would constitute a miscomputation, even if the external behaviour of the system were identical. On the other hand the mechanistic account does make reference to internal states (see Piccinini 2007: 501), and so we should not settle for an entirely external/behaviouristic theory of miscomputation. The correct answer, I think, is to treat certain internal components, such as logic gates, as minimal computational systems in their own right (*ibid*: 510). A malfunction in one of these components would then constitute a miscomputation, and even if the overall system continued to behave in the expected manner it could also be seen as miscomputing. A malfunction in a non-computational component, such as a fan, would not constitute a miscomputation unless the overall behaviour of the system was affected. This analysis of components as sub-mechanisms is advocated by Craver & Bechtel (2006), and is a central aspect of mechanistic explanation.

Behaviours described as “correct” or “expected” should not be understood as conforming to the designers intentions (what they thought they were doing), but rather as conforming to their actual design (what they did). For example, a designer might intend to write a program that translates one language into another, but fail and in fact write a program that produces gibberish. In this case there has been no miscomputation, as the program performs as designed, i.e. there is no mechanical malfunction. The only problem, from the designer's perspective, is that they did not produce the design that they intended to. A computing mechanism cannot be blamed for

"miscomputing" as a result of a design error, and in fact should not be said to miscompute at all in cases such as this.

Design errors are simply not relevant to the mechanistic characterisation of computation. The states of a computing mechanism are individuated non-semantically, but design errors "can only arise when some meaning is attached to the output signals from the machine" (Turing 1950: 449). There are many ways in which one might think an output signal comes to acquire meaning, but a central principle of the mechanistic account is that however this happens it cannot be a purely computational process (Piccinini 2007: 502). Design errors are essentially semantic phenomena, relying on the intentions of the designer, and thus, if the mechanistic account is correct, they should not qualify as genuine cases of miscomputation. This means that, at least with regard to whether or not a system is computing correctly, the mechanistic account only allows for analysis at the level of implementation. Higher levels of analysis might be appropriate for other purposes, such as producing a pragmatically useful design, but any mistakes that creep in prior to implementation cannot be considered strictly "computational" (in the mechanistic sense).

When a programmer writes an algorithm his intention might be that it do *x*, whilst in fact it does *y*. In this case when implemented the algorithm's function should be considered *y*, rather than *x*. The computing mechanism would only miscompute if it failed to carry out *y*, even if from the designers perspective it should be carrying out *x*. Design errors should be treated as a kind of user error rather than as genuine cases of miscomputation, in the sense that the failure is attributed to an external agent rather than to the computing mechanism. Such errors are more akin to a naïve user failing to operate a word processing program than a computing mechanism suffering from a mechanical malfunction. There are of course important differences between a designer and a naïve user, most obviously with regard to their knowledge of the system, but these differences are not relevant to the current topic. The point of the analogy is that in both cases it is the user/designer who is to blame for any errors that occur, not the computational system. For this reason we should not treat design errors as cases of miscomputation.

What is important when it comes to mechanistic miscomputation is the performance of the computational system in relation to its current design, regardless of how well that design conforms to the designer's intentions. A computing mechanism should be considered in isolation from the intentions of its designer, which may or may not have been correctly implemented. In assessing it's success or failure we should focus instead on its actual implementation at a certain point in time, and whether, given this implementation, it performs as expected. If it does not then there has been an operational malfunction, and we can genuinely say that it has miscomputed.

Towards the beginning of their article, Fresco & Primiero admit that "a computational

system *can only* make an error of *functioning* (i.e., an operational malfunction)" (2013: 3, emphasis in original). In this sense they acknowledge the point that I am making here, i.e. that considered in isolation a computing mechanism is incapable of performing a design error. They go on to include both types of errors (operational and design) as cases of miscomputation. Whilst at several points they indicate important differences between the two, I feel that referring to both as miscomputation risks conflating them in an unhelpful and potentially misleading manner. The point might well be terminological, but in this case the terminology is important, as it relates to deeper conceptions of what computation is, and how we should investigate it. The identification of a design error is reliant upon our knowledge of the agent who implemented the design, whilst operational errors can be identified simply by investigating the computational system in question. The term "miscomputation", which suggests the latter course of action rather than the former, should be reserved exclusively for operational errors.

I have argued that, according to the mechanistic account, miscomputation should be restricted to what Fresco & Primiero refer to as "operational" errors, i.e. errors that can be attributed solely to the computational system itself, without having to refer to the agent who designed the system. I feel that this restriction is of benefit to our understanding of computation, as it maintains a clear division between on one hand the purely mechanistic description of an isolated computational system (and of miscomputation), and on the other the semantically laden characterisation of a computational system's interaction with the external world (and of the resulting design errors). Making this division clear is a central motivation of the mechanistic account, albeit a somewhat controversial one (cf. Sprevak 2010). It restricts the analysis of miscomputation to the level of implementation, and more generally de-emphasises the idea that computation can be analysed at multiple levels. Piccinini (2007: 510-2) does make a distinction between abstract letters and concrete digits, but it is not clear that these are separate levels of analysis in the traditional sense. It must be admitted that this is an area in which more work could be done to clarify the mechanistic account.

If one felt that miscomputation should be understood in a broader sense, encompassing both internal and external features of a computational system, then the division described above could be seen as a weakness of the mechanistic account. Conversely, if one was opposed to the mechanistic account, then one might also be inclined to favour a broader taxonomy of miscomputation, along the lines of that presented by Fresco & Primiero. Either way, it is important to make the mechanistic account of miscomputation clear, in order that we may move forward with the important project that Fresco & Primiero have started.

References

- Craver, C. & Bechtel, W. (2006). Mechanism. In Sarkar & Pfeifer (eds.), *Philosophy of Science: an encyclopedia*: 469-78. New York: Routledge.
- Fresco, N. & Primiero, G. (2013). Miscomputation. *Philosophy & Technology* 26: 253-72.
- Piccinini, G. (2007). Computing Mechanisms. *Philosophy of Science* 74: 501-26.
- Sprevak, M. (2010). "Computation, individuation, and the received view on representation." *Studies in History and Philosophy of Science* 41: 260-270.
- Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind* 59 (236): 433-460.